CSE 589

Programming Assignment 2  
 Routing Protocols

Documentation  
 Of Code &   
 Report

By Anriudh Yellapragada  
 UBIT # 50206970

**Code Details** :  
All source code has been written into main.cpp.  
Data structure of the update message (Line Number) : 57- 66   
Data structure of the routing message (Line Number): 50 – 54

There is a makefile named as “makefile” which shall create an executable of name “server” **Assumptions** :  
The servers are named in increasing order of type integer.  
Infinity : 32000(Approximately equal to shrt\_max).

**All structures in the source file are listed as below :**

// Network Elements Information

struct NetworkElementsInfo {

short int id;

string ipAddress;

short int portNumber;

bool isNeighbour ;

};

// Current Elements Information

struct ThisElementInfo {

short int id;

string ipAddress;

short int portNumber;

friend ostream& operator << (ostream& os, const NetworkElementsInfo& m);

};

// Current element's neighbours

struct ThisElementNeighbourInfo {

short int id;

short int cost;

char ipAddress[INET\_ADDRSTRLEN];

short int portNumber;

friend ostream& operator << (ostream& os, const ThisElementNeighbourInfo& m);

};

struct RoutingTableElement {

short int destinationServerId;

short nextHopServerId;

short int costOfPath;

};

struct NeighbourDistanceVectorDetails {

short int numOfUpdateFields;

char ipAddress[4];

short int buffer[5];

short int portNumber;

short int neighbourIdArr[5];

short int costArr[5];

short int portnumberArr[5];

char ipAddressArr[5][4];

};

**Lists maintained :**list<NetworkElementsInfo> networkElementsList;

list<ThisElementNeighbourInfo> thisElementNeighboursList;

list<NeighbourDistanceVectorDetails> listOfNeighbourDistanceVectors;

ThisElementInfo thisElementInfo;

list<RoutingTableElement> routingTableList;  
  
**Few important points about the code** :   
1. We are maintaining a networkElementsList to store the total number of elements/servers in the network. This information is given by the topology file.  
2. We are maintaining a thisElementNeighboursList, to store the neighbours of the current element/server. This list has to be carefully modified For example, when we receive an update command to one of the neighbours as infinity, we remove the neighbour from this list. When we do not receive any distance vector update from one of the server’s neighbours for three consecutive intervals, then this neighbour should be removed from the list. When a disable command is given with a server id specified, this server id is no more a neighbour and hence should be removed from the list.  
3. We are maintaining a thisElementInfo struct, to store the ip address, id, port number of the current element/server. This is just to simplify the use of the current element’s id.  
4. We are maintaining a listOfNeighbourDistanceVectors, to store the distance vectors received from the neighbours. When a distance vector is received from the same neighbour after a particular time interval, then the existing distance vector we have is updated/modified based on the server id from which we receive the update.  
5. We are maintaining a routingTableList, to store the details like destination id, next hop server id and the least cost calculated using the Bellman Ford equation. This is a list of the destination id’s present in the network to which the leas cost and next hop has to be calculated.  
  
**Data structure of the update message:**  
Name : NeighbourDistanceVectorDetails  
As mentioned in the project specification document, the message format to be sent as a distance vector update is constructed/populated in this struct and then sent across to the neighbours. We are suppose to maintain a format of this message as per below mentioned details :  
Number of update fields: (2 bytes):Indicate the number of entries that follow.

- Server port: (2 bytes) port of the server sending this packet. (short int numOfUpdateFields)

- Server IP: (4 bytes) IP of the server sending this packet.( char ipAddress[4])

- Server IP address n: (4 bytes) IP of the nth server in its routing table.( char ipAddressArr[5][4])

- Server port n: (2 bytes) port of the nth server in its routing table.( short int portNumber)

- Server ID n: (2 bytes) server id of the nth server on the network.( short int neighbourIdArr[5];)

- Cost n: cost of the path from the server sending the update to the nth server whose ID is given in the packet.( short int costArr[5])

-Buffer n: An empty buffer space of 2 bytes for each server. (short int buffer[5])

**All functions in the source file are listed as below :**

1. main()

This function is hit when the command “./server.exe –t <topology\_file> -i <update\_interval> “ is given by the user.  
This function is used to invoke all the functions necessary to complete the project.

1. readTopologyFile()  
   This function reads the topology file and populates the below mentioned lists :
2. networkElementsList which is a list of elements of the type NetworkElementsInfo.  
   Here is the implementation :   
   list<NetworkElementsInfo> networkElementsList;
3. thisElementNeighboursList which is a list of elements of the type ThisElementNeighbourInfo

Here is the implementation :  
list<ThisElementNeighbourInfo> thisElementNeighboursList;

1. getMyIp()  
   This function is used to get the IP address of the current server/machine.  
   This IP is used to identify the server ID and the port number details from the topology file.
2. buildRoutingTable()  
   This function is used to build the routing table from the current information provided in the topology file.
3. sendDistanceVectorUpdate()  
   This function is used to populate/construct the message to be sent as a distance vector update to the neighbours of the current server.
4. select()  
   This function is used to monitor the socket created by the current process, the user inputs given by the user and the periodic update of the distance vector using timeout argument efficiently without blocking.
5. calculateDistanceVector()  
   This function is used to calculate the shortest distance to all the destinations from the current network element using Bellman Ford equation.
6. To handle various input commands that an user can give, we have implemented different functions as below :  
   a. update()  
   Command supported : update <server id1> <server id2> <cost>  
   This function is used to handle update command given by user. If the cost given is infinity, the server id2 is made as non-neighbour and the cost is updatded in costArray accordingly  
   **Implementation details:**

Here we create a UDP socket to send a temporary update message to the server id2 mentioned by the user command to indicate the modified cost.  
If the cost mentioned is “inf”, then the server id2 is made a non-neighbour and the costArray is updated with the maximum value.  
Now, we update the routing table as the costAray is modified, using Bellman Ford equation. If the cost mentioned is not “inf”, we send this calculated distance vector to the server id2.  
  
  
  
  
  
b. step()  
This function is to used to send distance vector update by force which is independent of the periodic update.  
**Implementation details:**

The implementation of this function is similar to sending a periodic update of distance vector which is constructing a message, creating a UDP socket and send the distance vector to all the neighbors’.  
c. packets()  
This function is used to display the number of packets received by the current network element as a part of distance vector update from its neighbors’ since the last instance this information was requested  
**Implementation details:**   
We maintain a global variable(say packet) and increment it whenever the current element received the distance vector update from its neighbors’.  
So when the packets command is given by the user, we display the value of the packet variable and make it zero. To achieve the requirement that this command should display number of packets received since the last instance this information was requested, we are making the packet variable as zero.  
d. display()  
This function is used to display the routing table at any instance of time.  
**Implementation details:**This function simply sorts the list of routing elements info based on the destination server id before displaying as per the requirement specified.  
e. disable()  
This function is used to disable the server id mentioned as a part of the command given by the user.  
Command supported : disable <server Id>  
**Implementation details:**

This function simply takes in the server id value given by the user and changes the costArray from the current element to the server id. The server id is also made non-neighbor. The routing table is updated accordingly.  
f . crash()  
This function is used to emulate a crash and set the link cost to infinity to all its neighbors’  
**Implementation details:**This function simple closes all the sockets that have been created during the distance vector update and then exits. The neighbors’ of the server which has been crashed will identify that this neighbour has crashed as they won’t receive the distance vectors for more than three consecutive intervals from the crashed sever.

**REPORT:**

The distance vector is calculated using the Bellman-Ford algorithm. The code flow before calculating the equation is as follows.  
General base case :  
1. The neighbour’s message has to be parsed.   
2. costArray is to be modified.  
3. calculateDistanceVector() – To calculate Bellman Ford equation.  
Update command :  
1. Parse the update command to get the updated cost to a neighbour.  
2. Change the cost array.  
3. Send a UDP packet to the speicified neighbor regarding the updated cost.

4 .Calculate Distance Vector.  
Disable command:  
1. Parse the disable command to get the server id.  
2. Change the cost array to make the distance as infinity.  
3. Calculate Distance Vector.

**When links are updated:**

When a link in the network is updated then the neighbor will not immediately get to know about the change unless and until it has been informed about the change. The neighbor then calculates its new distance vector. Before this calculation is done, as the system inherently is designed to send periodic updates rather than an update when modification of distance vector is occurred, it ends up sending wrong distance vectors to its neighbours’ for a certain amount of time(Time of convergence).   
This bad updates will continue till they converge and the time taken for the convergence to be achieved is directly proportional to the cost which is updated leading the system to go into a bad state initially.

**Count to Infinity :**

**Base Case :**

Disabling a link or a crash of one of the routers in the network can lead to worse conditions. An important recovery system that we can adopt here is that this crash update(that a server has gone done) has to be communicated very quickly across the network by its neighbor. If this message is not sent across to all the servers in the network in proper time it might result in a bad condition where one router misleads the other router about the existence of a non-existing path. This ultimately results in a routing loops leading to count to infinity problem.

**Timeout Case :**   
In this project, we are actually explicitly inducing the count to infinity by checking for the 3 consecutive update intervals, where an update has been received or not. This leads to a change in the cost array to infinity which means that the network element is no more a neighbor. This implies that we implicitly calculate, as in the update case, the costs based on this value which is a false positive.

**Update Case :**  
In this project, as an user can perform update command, we are bound to accept an update in the network topology. If the update of the cost in a value greater than the previous cost, this leads to count to infinity till some point called time of convergence. This is a kind of count to infinity induced by the user.  
  
**Conclusion :**   
A routing protocol which has been implemented based on the Distance Vector Algorithm is a better protocol when compared to the protocol implemented using the link state algorithm. This is because the link state algorithm is a centralized algorithm whereas, the distance vector algorithm is an iterative, decentralized, dynamic and distributed algorithm. The only disadvantage of this dynamic algorithm is that inherently suffers from count-to-infinity problem.